HW Assignment #5

Oracle PL/SQL

stored functions/procedures, constraints/triggers

KAIST

Prof. Myoung Ho Kim

Contents

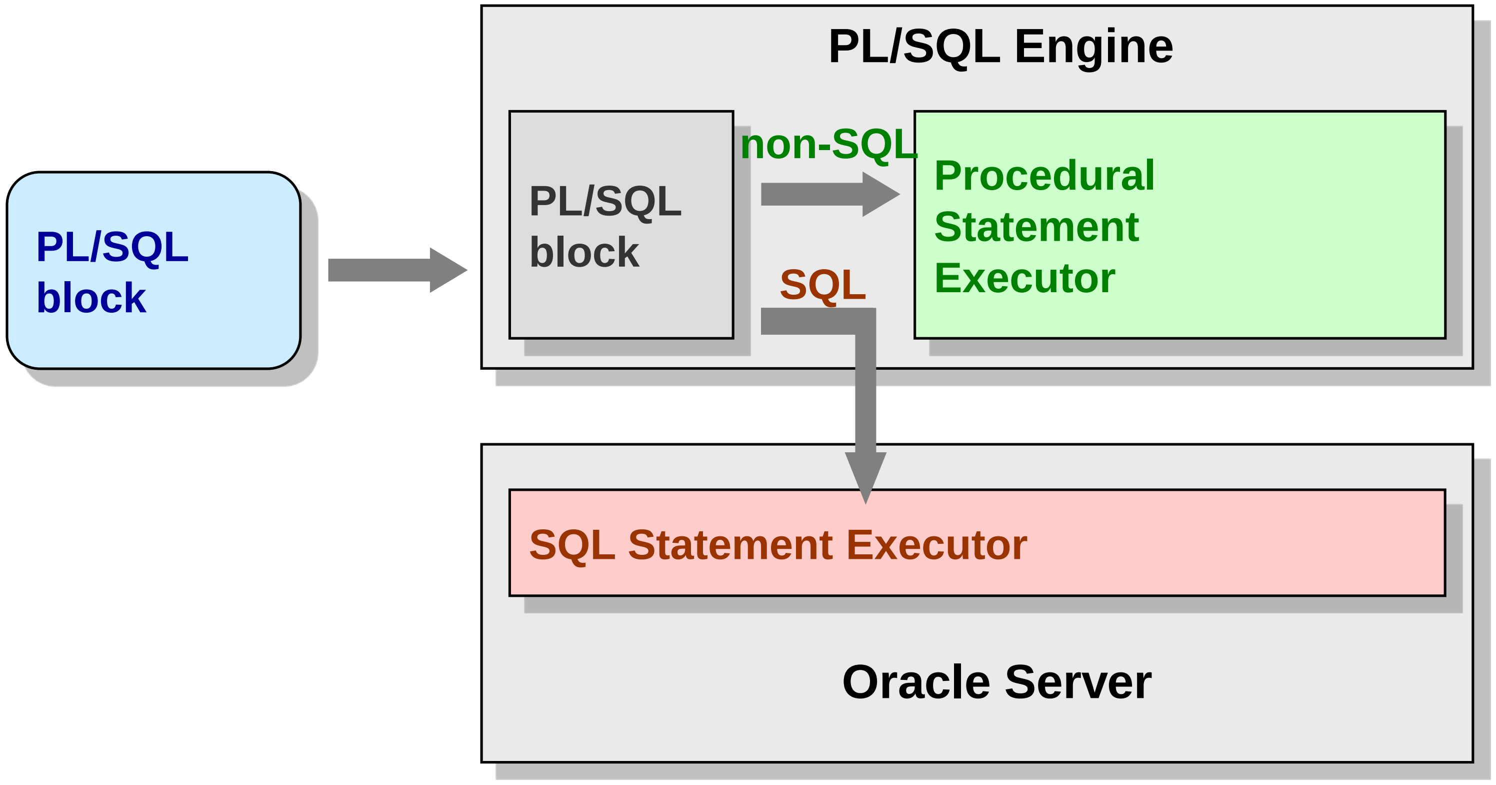
- PL/SQL
  - Introduction to PL/SQL
  - Stored procedures/functions
  - Variables and their types
  - Control structures
  - Cursors
  - Exception handling
  - Triggers
- Calling stored procedures/functions in JDBC
- Homework Assignment #5

PL/SQL: Introduction

- PL/SQL (Procedural Language-SQL)
  - Procedural SQL Language
    - Oracle Corporation's procedural extension language for SQL to include the properties in programming languages
    - » Modularize the program development
    - » Support variable declaration
    - » Support loop and conditional statement
    - » Support exception handling
  - Enhance the execution performance
    - » Execute many SQL statements at once

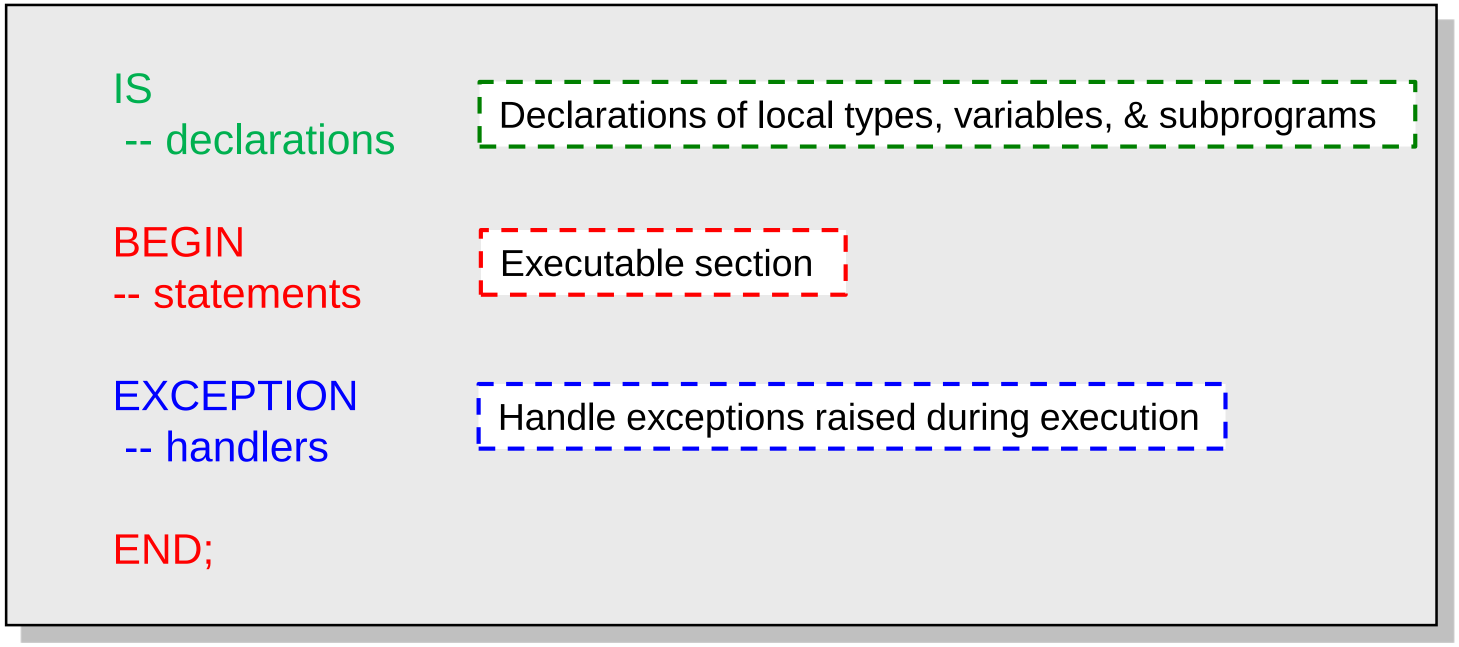
PL/SQL: Introduction

- PL/SQL environment



PL/SQL: Introduction

- PLSQL Block Structure



```
IS
 -- declarations

BEGIN
-- statements

EXCEPTION
 -- handlers

END;
```

Declarations of local types, variables, & subprograms
Executable section
Handle exceptions raised during execution

PL/SQL: Introduction

- PL/SQL block
  - PL/SQL subprogram
      - ***named PL/SQL block***
    - Stored procedures and stored functions
      - stored in database and can be called repeatedly
      - functions return a result
    - Triggers
      - stored subprogram associated with a table, view, or event
      - invoked when specific events occur
        - e.g. INSERT, DELETE, UPDATE
  - Anonymous block
    - make and execute whenever needed
    - not stored in the database

PL/SQL: Stored Procedures

- Creation

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(argument1 [mode] data_type1,
  argument2 [mode] data_type2,
  ......)]
IS
(variable declarations)
BEGIN
(code for execution)
[EXCEPTION]
(exception handling)
END;
```

mode = [IN|OUT|INOUT]

To see compile errors
Type ***SHOW ERRORS***
in SQL-Plus

- Deletion

```
DROP PROCEDURE procedure_name;
```

PL/SQL: Stored Procedures

- How to execute stored procedures
  - In SQL*Plus: use EXECUTE statement

```
SQL> EXECUTE update_grade( 70541 );
```

  - In procedures: write the procedure's name to be called within BEGIN, END clause

```
BEGIN
    update_grade( 70541 );
END;
```

"CALL" is not needed

  - In applications (C, Java etc):

```
Connection con = DriverManager.getConnection(strConn, "s20150000", "s20150000");
CallableStatement cs = con.prepareCall("{call update_grade(?)}");
cs.setInt(1, 70541);
cs.executeUpdate();
```

Example with JDBC

If cannot execute the stored procedure in console,
You type "**show serveroutput**" and "**set serveroutput on**"
when the status is off

(EX) Stored Procedure

procedure.sql

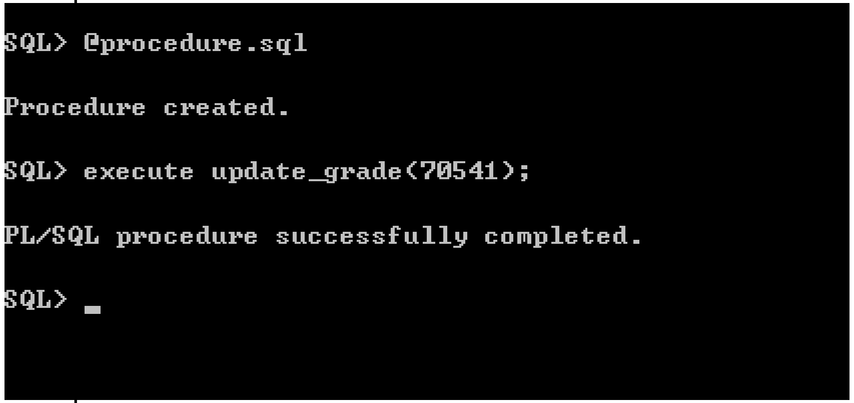
```
CREATE OR REPLACE PROCEDURE update_grade
(v_sid IN NUMBER)

IS

BEGIN
      UPDATE ScoreRecord
      SET Score= Score + 0.3
      WHERE Studentid = v_sid AND
               courseID = ‘CS310’;
END;
/
```

‘/’ in the last line means
“execute CREATE statement”

Execute in SQL*Plus



```
SQL> @procedure.sql

Procedure created.

SQL> execute update_grade(70541);

PL/SQL procedure successfully completed.

SQL>
```

PL/SQL: Stored Functions

- Creation

```
CREATE [OR REPLACE] FUNCTION function_name
[(argument1 [mode] data_type1,
  argument2 [mode] data_type2,
  ......)]
RETURN data_type
IS
(variable declarations)
BEGIN
(code for execution)
RETURN (value);
[EXCEPTION]
(exception handling)
END;
```

- Deletion

```
DROP FUNCTION function_name;
```

PL/SQL: Stored Functions

- How to execute
  - In SQL*Plus:
    - » Declare a bind variable to save the return value
    - » type EXECUTE statement(to see the return value, use PRINT statement)

```
SQL> VARIABLE score NUMBER;
SQL> EXECUTE :score := get_score(71041);
```

  - In procedures:

```
    score NUMBER
BEGIN
    score := get_score(71041);
END;
```

  - In applications (C, Java etc):

```
Connection con = DriverManager.getConnection(strConn, “s20150000", “s20150000");
CallableStatement cs = con.prepareCall("{? = call get_score(?)}");
cs.registerOutParameter(1, Types.NUMERIC);
cs.setInt(2, 74041);
cs.execute();
int score = cs.getInt(1);
```

Example with JDBC

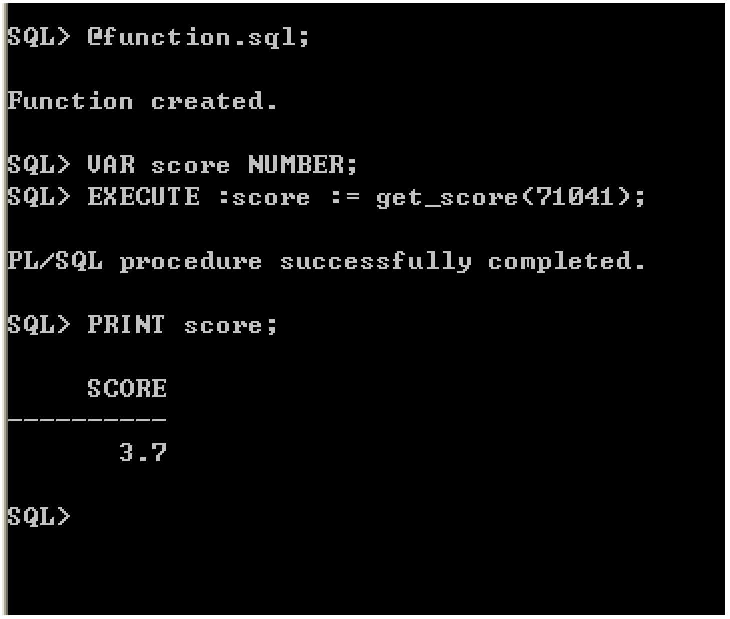
PL/SQL: Example - stored function

function.sql

```
CREATE OR REPLACE FUNCTION get_score
(v_sid IN NUMBER)
RETURN NUMBER
IS
   v_score ScoreRecord.Score%TYPE;
BEGIN
   SELECT Score INTO v_score
      FROM ScoreRecord
      WHERE StudentId = v_sid AND
                 CourseID = ‘CS360’;
   RETURN v_score;
END;
/
```

‘/’ in the last line means “execute CREATE statement”

Execute in SQL*Plus



```
SQL> @function.sql;

Function created.

SQL> VAR score NUMBER;
SQL> EXECUTE :score := get_score(71041);

PL/SQL procedure successfully completed.

SQL> PRINT score;

     SCORE
----------
       3.7

SQL>
```

PL/SQL: Variables

- Declaration
  - Declare in IS

```
variable_name [CONSTANT] data_type [NOT NULL] [:=
value];
```

Example

```
v_empno NUMBER := 0;
```

- Assign a value

```
variable_name := value or expression;
```

Example

```
v_price := 5000;
tax := price * tax_rate;
amount := TO_NUMBER(SUBSTR(‘750 dollars’, 1, 3));
```

PL/SQL: Variable types

Supplement
- *maybe convenient if you use it*

- Overview of PL/SQL datatypes
  - Scalars
  - Collections
  - References
    - %TYPE, %ROWTYPE

PL/SQL: Variable types

Supplement
- maybe convenient if you use it

- Scalars: for assigning single value
  - NUMBER, BINARY_INTEGER, CHAR, VARCHAR2, LONG, LONG RAW, DATE, BOOLEAN
- Collections: for assigning multiple values
  - Associative **arrays**
    - » different concept from the **Table** in database

```
TYPE enames_type IS TABLE OF varchar2(10)
   INDEX BY NUMBER;
enames    enames_type;
……
BEGIN
   enames(0012) := ‘jhseo’;
   ……
END;
```

INDEX BY VARCHAR2(10)
is also possible
e.g. enames(‘cs360_TA’) := ‘jhseo’;

PL/SQL: Variable types

Supplement
- maybe convenient if you use it

- Collections (cont'd)
  - Records
    - » A collection of fields
    - » similar to structure type in C

```
TYPE dept_record_type IS RECORD
   (deptno     NUMBER(2),
     dname    VARCHAR2(13),
     loc         VARCHAR2(14));
dept_record    dept_record_type;
……
BEGIN
   dept_record.deptno := 10;
   ……
END;
```

PL/SQL: Variable types

Supplement
- maybe convenient if you use it

- References
  - %TYPE
    - » *variable_name table_name.col_name*%TYPE
      - Refer to the datatype of *table_name.col_name*

```
v_name      Student.Name%TYPE;
v_score     ScoreRecord.Score%TYPE;
```

declare

variable	datatype
v_name	VARCHAR2(15)
v_score	NUMBER

result

PL/SQL: Variable types

Supplement
- maybe convenient if you use it

- References (cont'd)
  - %ROWTYPE
    - » Contain the information of a row in a specific table
    - » Usage
      - *variable_name table_name*%ROWTYPE

deptno	dname	loc
9	EE	Daejeon
:	:	:

dept **table**

```
v_dept   dept%ROWTYPE;
……
BEGIN
   v_dept.deptno := 10;
   v_dept.dname := 'CS';
   v_dept.loc := 'Daejeon';
   ……
END;
```

PL/SQL: Control structures

- Conditional control (IF, END IF)

Syntax

```
IF condition1 THEN …
[ELSEIF condition2 THEN …]
[ELSE …]
END IF
```

Example

```
BEGIN
        IF sales > 50000 THEN
            bonus := 1500;
        ELSEIF sales > 35000 THEN
            bonus := 500;
        ELSE
            bonus := 100;
        END IF;
END;
```

PL/SQL: Control structures

- Iterative control (LOOP, END LOOP)

Syntax

```
LOOP
        sequence of statements
        [EXIT WHEN condition]
END LOOP
```

Example

```
DECLARE
v_cnt NUMBER(3) := 100;
BEGIN
LOOP
        INSERT INTO emp (empno, ename, hiredate)
              VALUES (v_cnt , 'Tom', sysdate);
        v_cnt := v_cnt + 1;
        EXIT WHEN v_cnt > 110; -> exit condition
END LOOP;
END;
/
```

PL/SQL: Control structures

Supplement
- maybe convenient if you use it

- Iterative control (FOR, END LOOP)

Syntax

```
FOR counter IN [REVERSE] min_value .. max_value LOOP
    sequence_of_statements
END LOOP
```

NOTE:
We also can use CURSOR in FOR-LOOP

Example

```
DECLARE
i NUMBER;
BEGIN
FOR i IN 1 .. order_qty LOOP
            UPDATE sales SET custno = customer_id
            WHERE serial_num = serial_num_seq;
END LOOP;
END;
```

PL/SQL: Control structures

Supplement
- maybe convenient if you use it

- Iterative control (WHILE, END LOOP)

Syntax

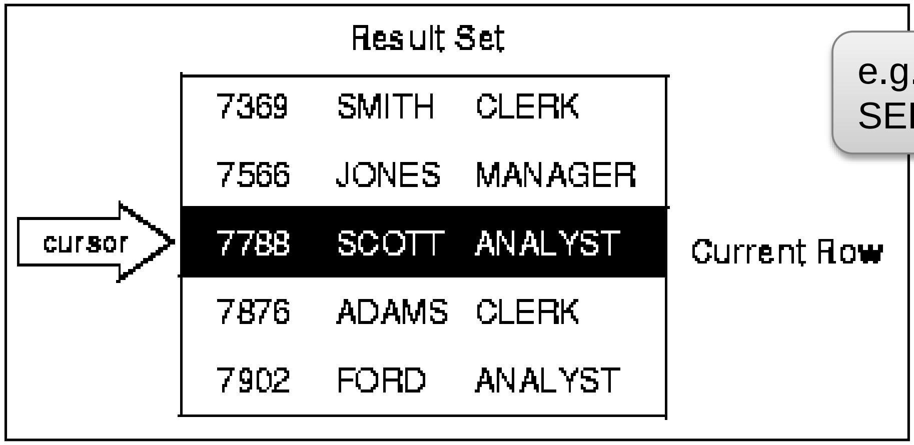
```
WHILE condition LOOP
   sequence_of_statements
END LOOP
```

Example

```
WHILE total < 25000 LOOP
         …
         SELECT sal INTO salary FROM emp WHERE …
         total := total + salary;
END LOOP;
```

PL/SQL: Cursors

- Workspace
  - Oracle saves the execution result of SQL in the workspace
    - » For DML: whether the statement succeeds
    - » For SELECT: result tuples
- Cursor
  - Used to access the data in the workspace



7369	SMITH	CLERK
7566	JONES	MANAGER
7788	SCOTT	ANALYST
7876	ADAMS	CLERK
7902	FORD	ANALYST

e.g. access the result tuples of SELECT statement

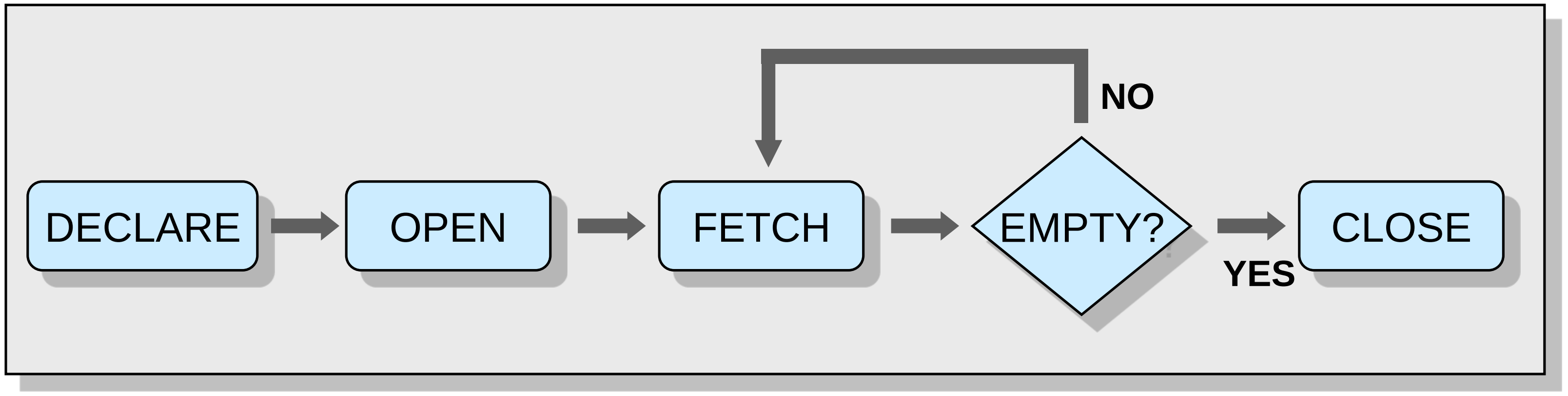
PL/SQL: Implicit Cursors

Supplement
- maybe convenient if you use it

- Automatically generated by DML(e.g., UPDATE) and SELECT statements
  - If there is SQL statement in BEGIN SECTION, <u>a cursor named SQL</u> is automatically(implicitly) generated
- Attributes
  - SQL%ROWCOUNT
    - » the number of ROWs affected by the most recent SQL statement
  - SQL%FOUND
    - » TRUE if the most recent SQL statement returns one or more rows
  - SQL%NOTFOUND
    - » TRUE if the most recent SQL statement does not return any tuple
  - SQL%ISOPEN:
    - » Check if the cursor is open
    - » NOTE
      - Oracle closes the SQL cursor automatically after executing its associated SQL statement. As a result, %ISOPEN is always yields FALSE

PL/SQL: Explicit Cursors

- Named cursors
  - Declared by users(PL/SQL programmers)
  - 4 steps
    - DECLARE, OPEN, FETCH, and then CLOSE



PL/SQL: How to use explicit cursors

- DECLARE
  - Write a SELECT statement to be executed

```
CURSOR cursor_name IS [SELECT_statement];
```

- OPEN
  - Execute the *SELECT statement* defined in DECLARE

```
OPEN cursor_name;
```

PL/SQL: How to use explicit cursors

- FETCH
  - Read a result tuple of the *SELECT statement*
  - If there are multiple tuples, use iterative controls

```
LOOP
        FETCH cursor_name INTO variable;
        EXIT WHEN condition;
END LOOP;
```

→ Check if any remaining tuple exists

- CLOSE
  - Release the specified cursor

```
CLOSE cursor_name;
```

PL/SQL: How to use explicit cursors

- Attributes: followed by the cursor name
    - » e.g. *cursor_name%attribute*
  - – %ROWCOUNT
    - » The number of rows FETCHed so far
  - – %FOUND
    - » TRUE if the last FETCH returned a row
  - – %NOTFOUND
    - » TRUE if the last FETCH failed to return a row
  - – %ISOPEN
    - » TRUE if the cursor is open

(EX) Explicit Cursor

```
CREATE OR REPLACE PROCEDURE emp_process
IS
        v_empno  emp.empno%TYPE;
        v_ename  emp.ename%TYPE;
        v_sal       NUMBER;
        CURSOR emp_cursor IS
            SELECT empno, ename, sal
            FROM emp
            WHERE deptno = 20;
BEGIN
        OPEN emp_cursor;
        LOOP
                FETCH emp_cursor INTO v_empno, v_ename, v_sal;
                   …
                EXIT WHEN emp_cursor%NOTFOUND;
        END LOOP;
        CLOSE emp_cursor;
END;
/
```

PL/SQL: Exceptions

- Exceptions
  - In PL/SQL an error condition is called “exception”
    - » e.g. ZERO_DIVIDE, STORAGE_ERROR
  - When an error occurs, an exception is raised
    - » i.e. normal execution stops and control transfers to the exception-handling part of the PL/SQL block
  - When is an exception raised?
    - » System error occurs
    - » PL/SQL code calls “RAISE” statement

PL/SQL: Predefined Exceptions

Supplement
- maybe convenient if you use it

- **Predefined PL/SQL exceptions**
  - **named and unnamed exceptions**
    - » 21 system errors have predefined names
    - » other exceptions have their error code but have no name

Exception names	Description
INVALID_CURSOR	Attempts to a cursor operation not allowed (e.g., closing an unopened cursor)
ZERO_DEVIDE	Attempts to divide a number by 0
DUP_VAL_ON_INDEX	Attempts to store duplicate values in a column has UNIQUE constraint
NOT_LOGGED_ON	Issues database call without being connected to Oracle database
…	…

PL/SQL: User Defined Exceptions

- Declaration
  - Declare in IS SECTION

```
exception_name EXCEPTION;
```

- Usage
  - Raise an exception with “RAISE” statement in BEGIN SECTION

```
RAISE exception_name;
```

PL/SQL: Exception Handling

- EXCEPTION SECTION

OTHERS handler catches all exceptions that the block does not name specifically

```
EXCEPTION
    WHEN exception_name1 [OR exception_name2 ...] THEN
        sequence_of_statements;
        ......
    [WHEN exception_name3 [OR exception_name4 ...] THEN
        sequence_of_statements; ......]
    [WHEN OTHERS THEN
        sequence_of_statements; ......]
```

  – RAISE_APPLICATION_ERROR(*error_number*, *message*)
    - » Define user's own error message that printed in stdout
    - » *error_number* a negative integer in the range -20000 ~ -20999 and *message* is a character string up to 2048 bytes long

"Please type valid phone number" is more meaningful rather than "ORA-02290: Check constraint violation error"

(EX) Exception Handling

```
CREATE OR REPLACE PROCEDURE User_Exception
(v_deptno IN emp.deptno%type)
IS
        user_defined_error EXCEPTION;
        cnt NUMBER;
BEGIN
        SELECT COUNT(empno) INTO cnt
           FROM emp WHERE deptno = v_deptno;
        IF cnt < 5 THEN
           RAISE user_defined_error;
        ENDIF;
EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
           RAISE_APPLICATION_ERROR(-20000, ‘It’s already in the table’’);
        WHEN user_defined_error THEN
           RAISE_APPLICATION_ERROR(-20001, ‘The department has too low employees’);
END;
```

(EX) Get Error Code in JDBC

```
try{
   …
   String strConn
             = “jdbc:oracle:thin:@dbclick.kaist.ac.kr:1521:orcl”;
   con = DriverManager.getConnection(strConn, “s20150000", “s20150000");
   cs = con.prepareCall("{? = call func_update_sal(?)}");
   cs.registerOutParameter(1, Types.NUMERIC);
   cs.setInt(2, 30);
   cs.execute();
   …
} catch(SQLException e) {
   int i = e.getErrorCode();             //if ORA-02290 occurs, i is 2290
}
```

PL/SQL: Exception Handling

- Handling unnamed exceptions
  - » exceptions that have only system error codes
  - » e.g. CHECK constraint violation

1. Use OTHERS handler
2. Give a name to the exception

- Give names to the unnamed exceptions
  - – Declare user-defined exceptions
  - – Assign the exception names to the unnamed error numbers
    - » PRAGMA EXCEPTION_INIT(*user-defined exception*, *error_number*)

Error number that returned by the system

(EX) Handling Unnamed Exception

ORA-02290: Check constraint violation error

```
CREATE OR REPLACE PROCEDURE Exception_Naming
IS
     check_violated EXCEPTION;
     PRAGMA EXCEPTION_INIT(check_violated, -2290);
BEGIN
     …
EXCEPTION
     WHEN check_violated THEN
          RAISE_APPLICATION_ERROR(-20001, ‘It’s invalid value’);
END;
```

PL/SQL: Triggers

- Event-driven PL/SQL subprogram
  - Database calls a trigger when a specific DML on a table is executed

- Elements
  - TIMING
    - » BEFORE, AFTER: when to execute the trigger (before or after executing DML statement)
  - TRIGGER_EVENT
    - » INSERT, UPDATE, DELETE: execute the trigger when INSERT, UPDATE, DELETE events occur
  - LEVEL
    - » STATEMENT: execute the trigger once
    - » ROW: execute the trigger for each row affected by the DML statement

PL/SQL: Triggers

- Creation

```
CREATE [OR REPLACE] TRIGGER trigger_name
TIMING TRIGGER_EVENT1 [OR TRIGGER_EVENT2 ...]
[OF column_name] ON table_name
[REFERENCING
[NEW AS new_row_name][OLD AS old_row_name]]
[FOR EACH ROW] [WHEN (condition)]
DECLARE
(variable declarations)
BEGIN
(PL/SQL code for execution)
END;
```

TIMING = BEFORE | AFTER

$TRIGGER\_EVENT_x$ = INSERT | UPDATE | DELETE

- Deletion

```
DROP TRIGGER trigger_name;
```

PL/SQL: Triggers

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT OR UPDATE OR DELETE ON s_emp
BEGIN
        IF (TO_CHAR(SYSDATE, ‘DY’) IN (‘SAT’, ‘SUN’)) OR
           (TO_CHAR(SYSDATE, ‘HH24’) NOT BETWEEN ’09’ AND ’16’)
        THEN
           RAISE_APPLICATION_ERROR(-20201, ‘Unavailable time’);
        END IF;
END;
```

```
CREATE OR REPLACE TRIGGER prod_update
AFTER UPDATE OF dscp ON product
FOR EACH ROW
WHEN new.price < old.price * 1.5
BEGIN
        UPDATE order_details
        SET p_dscp = :old.dscp;
        WHERE p_id = :old.p_id;
END;
```

PL/SQL: Triggers

- Variable for referring events
  - FOR EACH ROW
    - » In BEGIN SECTION
      - :old – the row before being updated
      - :new – the row after being update
    - » In WHEN condition
      - Use old, new (no preceding colon!!)

$$\begin{tikzcd}
A \arrow[r] & B \\
:\mathrm{old} \arrow[u] & :\mathrm{new} \arrow[u]
\end{tikzcd}$$

  - Statement-level triggers(when FOR EACH ROW is omitted)
    - » These triggers have no variables for referring events
    - » The differences between SQL standard and Oracle triggers (refer to the following URL)
      - http://www-db.stanford.edu/~ullman/fcdb/oracle/or-triggers.html

Calling Stored Procedures/Functions in JDBC

Supplement

- Connection.prepareCall(String *escapeSyntax*)
  - Create an instance of *CallableStatement*
  - *escapeSyntax*: syntax for executing stored procedures and functions
  - *escapeSyntax* syntax
    - » {call *procedure_name* [(*parameter1*, *parameter2*, ...)]}
    - » {? = call *function_name* [(*parameter1*, *parameter2*, ...)]}

- *CallableStatement* interface
  - An interface for executing stored procedures/functions in Java (inherits *PreparedStatement* interface)

Supplement

Calling Stored Procedures/Functions in JDBC

- The main methods of *CallableStatement* interface

Method	Description
registerOutParameter(int parameterIndex, int sqlType)	Register an OUT parameter at the position *parameterIndex* with type *sqlType*
setXXX(int parameterIndex, XXX x)	Set a value *x*, type *XXX*, at the position parameterIndex
executeUpdate(), **execute**()	Execute a stored procedure or function
getXXX(int parameterIndex)	Return a value of type *XXX* at the position *parameterIndex* *Use this method when getting a result after executing a stored function*

- XXX: Int, String, Double etc.
- Every index values start with 1

registerOutParameter used for the ***OUT parameters*** of the stored procedures and the ***return*** value of the stored functions

(EX) Execute a Stored Procedure in JDBC

Supplement

```
Connection con = null;
CallableStatement cs = null;
try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch (ClassNotFoundException e) {}
try {
    String strConn
        = “jdbc:oracle:thin:@dbclick.kaist.ac.kr:1521:orcl”;
    con = DriverManager.getConnection(strConn, “s20150000", “s20150000");
    cs = con.prepareCall("{call update_sal(?)}");
    cs.setInt(1, 30);
    cs.execute();
}
...
```

(EX) Execute a Stored Function in JDBC

Supplement

```
String strConn
            = “jdbc:oracle:thin:@dbclick.kaist.ac.kr:1521:orcl”;
con = DriverManager.getConnection(strConn, “s20150000", “s20150000");
cs = con.prepareCall("{? = call func_update_sal(?)}");
cs.registerOutParameter(1, Types.NUMERIC);
cs.setInt(2, 30);
cs.execute();
int cnt = cs.getInt(1);
System.out.println(cnt);
```

Homework Assignment #5

Homework #5

- Library management system
  - Develop an application managing loan and return of books
  - Environment
    - Database: Oracle 11g
    - Programming language: PL/SQL and Java

Homework #5 (cont'd)

- Table Schema
  - BookInfo: table storing book information
    - » bid: book id
    - » Title: a title of a book
    - » nrLoaned: the number of times that the book has been loaned
  - Members: table storing member information
    - » mid: member id (e.g., student number)
    - » mname: member name
    - » type: a type of a member (e.g., “undergraduate”, “graduate”)
  - Loan: table storing currently loaned book records
    - » mid: loaner id
    - » bid: loaned book id
    - » dueDate: due date of a book
    - » nrExtension: the number of extension

Homework #5 (cont'd)

- Problem 1: Table creation with constraints
  - Create tables in the previous page with the following constraints
  - Data type and constraints (Primary key is underlined)
    - » BookInfo
      - <u>**bid**</u> : NUMBER
      - title: VARCHAR(128), **NULL is not allowed**
      - nrLoaned: NUMBER
    - » Members
      - <u>**mid**</u>: NUMBER
      - mname: VARCHAR(32), **NULL is not allowed**
      - type: VARCHAR(15), **only "undergraduate", "graduate", or "Professor"**
    - » Loan
      - <u>**mid**</u>: NUMBER, **refers to mid in Members table**
      - <u>**bid**</u>: NUMBER, **refers to bid in BookInfo table**
      - dueDate: DATE, **NULL is not allowed**
      - nrExtension: NUMBER, **0 <= nrExtension <= 3**

Homework #5 (cont'd)

- Problem 1 (cont'd): Virtual view creation
  - BookLoanInfo(bid, title, dueDate)
    - » Join BookInfo and Loan tables (bid and title from BookInfo, dueDate from Loan)
    - » Consists of book information and their due date
      - If a book is not loaned, dueDate=NULL

bid	title	…
1	Database Systems	
2	Intro. to Algorithm	
3	The kite Runner	

BookInfo table

Mid	bid	dueDate	…
20121234	1	2015/05/21	
20151231	3	2015/05/14	

Loan table

bid	title	dueDate
1	Database Systems	2015/05/21
2	Intro. to Algorithm	NULL
3	The kite Runner	2015/05/14

BookLoanInfo view

Homework #5 (cont'd)

- Problem 2
  - Make a **trigger**, *Trigger_loan_book,* for a loan process
    - » If a new record is added to Loan table (a book is loaned), then *nrLoaned* of the loaned book in BookInfo is increased by 1

Homework #5 (cont'd)

- Problem 3
  - Make a **stored function** that determines whether a user is a delinquent or not
    - » Functions
      - Input parameter: member id
      - Return value: 1 if he/she is a delinquent, 0 otherwise
      - If a user has loaned a book and due date has passed, then he/she becomes a delinquent
        - If TRUNC(SYSDATE – Loan.DueDate) > 0, then delinquent
    - » Exception
      - When the member is not in MEMBERS table

Homework #5 (cont'd)

- Problem 4
  - Make a **stored function** that estimates the late fee
    - » Functions
      - Input parameter: Book ID
      - Return value: late fee
      - Late fee: 0 if the book is not overdue. 100 per day if the book is overdue
    - » Exception
      - When the book is not in BookInfo table

Homework #5 (cont'd)

- Problem 5
  - Make a **stored procedure** that handles the book loan process by using processes and functions in previous problems
    - » Functions
      - Input parameter: (member id, book id)
      - Insert a new record into Loan table
      - Setting due date
        - Undergraduate student: after 7days of sysdate
        - Graduate student, professor: after 14 days of sysdate
    - » Exceptions
      - When the book is not in BookInfo table
      - When the book is already loaned
      - When the user is a delinquent

Homework #5 (cont’d)

- Problem 6
  - Make a **stored function** that handles the book return process by using processes and functions in previous problems
    - » Functions
      - Input parameter: book id
      - Delete the record of returned book in Loan table
      - Return value: **late fee** of the returned book
    - » Exception
      - When the book is not in Loan table

Homework #5 (cont'd)

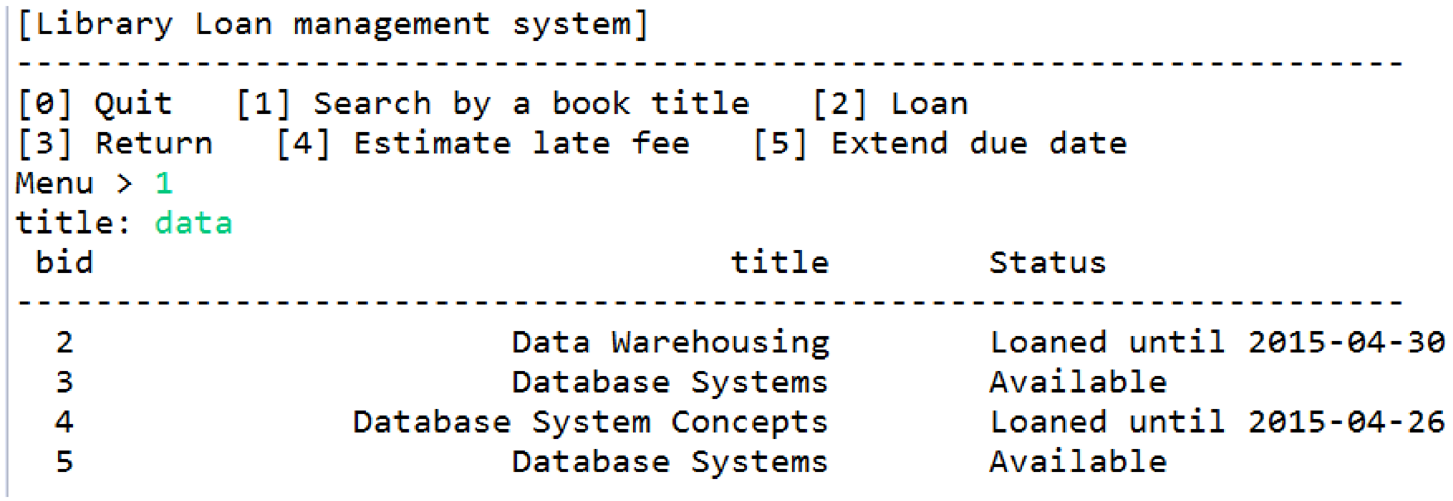
- Problem 7
  - Make a **stored procedure** for requesting due extension
    - » Functions
      - Input parameter: member id, book id
      - Due date is extended by 7 days
      - Update LOAN table – set new dueDate, and increase NrExtension by 1
    - » Exception
      - When there is no loan history with the member and the book
      - When the user is a delinquent
      - When the due date is already extended three times
        - Change "ORA-02290: check constraint error" to user-defined exception whose message is "Can't extend the loan more than 3 times"

Homework #5 (cont'd)

- Problem 8
  - Make a **Java program** that runs "Library management system" with "CallableStatement"
  - Java application should have following functions
    - » **1. Search by title (Using "PreparedStatement")**
    - » **2. Loan**
    - » **3. Return**
    - » **4. Estimate late fee**
    - » **5. Extend due date**
  - If an exception is raised, then print the error message
    - » Program must not be terminated by the exception

Homework #5 (cont'd)

- Example
  - Main menu & Search by a book title



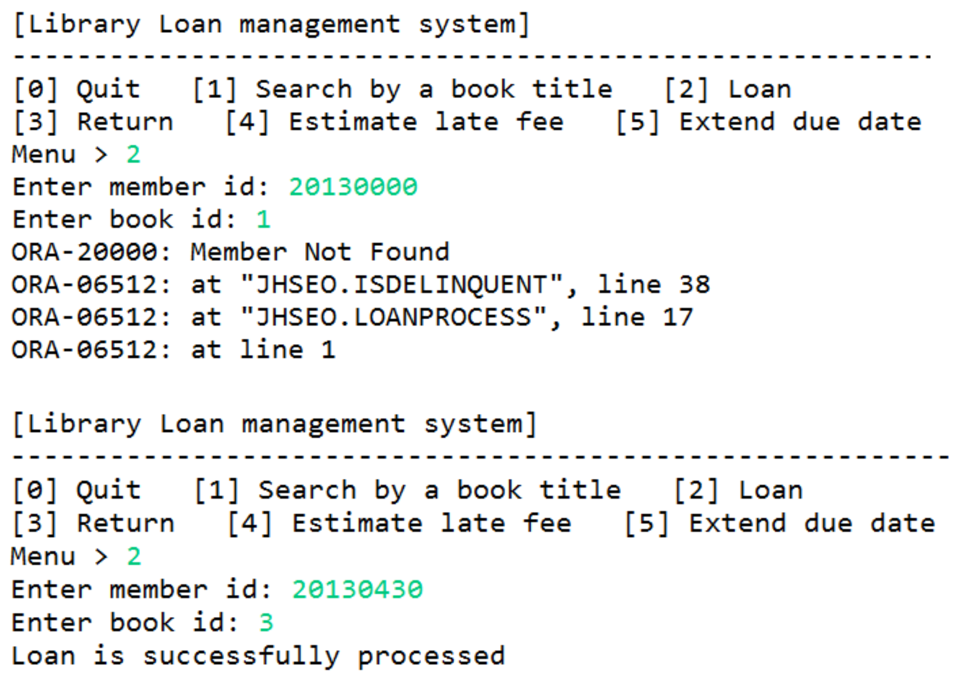
```
[Library Loan management system]
-----------------------------------------------------------------------
[0] Quit   [1] Search by a book title   [2] Loan
[3] Return   [4] Estimate late fee   [5] Extend due date
Menu > 1
title: data
 bid                                title        Status
-----------------------------------------------------------------------
  2                      Data Warehousing        Loaned until 2015-04-30
  3                      Database Systems        Available
  4               Database System Concepts        Loaned until 2015-04-26
  5                      Database Systems        Available
```

» Select records from BookLoanInfo view

- Query: "SELECT * FROM BookLoanInfo WHERE title LIKE %**Input**% ORDER BY bid"
- If BookLoanInfo.dueDate is null, then print "Available"
- Otherwise, print "Loaned until [**due date of books**]"

Homework #5 (cont'd)

- Example



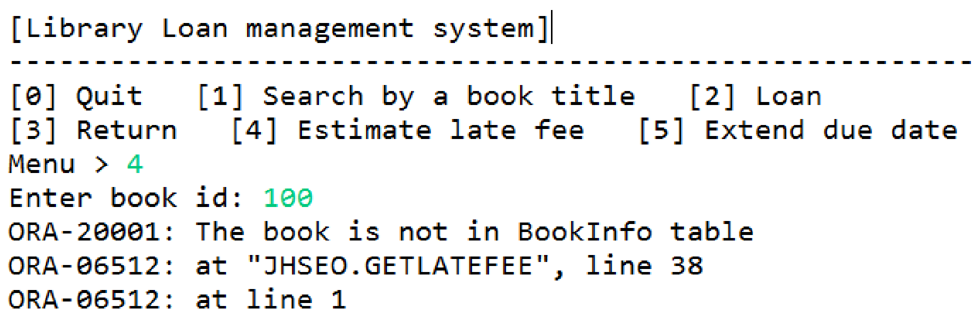
```
[Library Loan management system]
----------------------------------------------------------
[0] Quit   [1] Search by a book title   [2] Loan
[3] Return   [4] Estimate late fee   [5] Extend due date
Menu > 2
Enter member id: 20130000
Enter book id: 1
ORA-20000: Member Not Found
ORA-06512: at "JHSEO.ISDELINQUENT", line 38
ORA-06512: at "JHSEO.LOANPROCESS", line 17
ORA-06512: at line 1

[Library Loan management system]
-----------------------------------------------------------
[0] Quit   [1] Search by a book title   [2] Loan
[3] Return   [4] Estimate late fee   [5] Extend due date
Menu > 2
Enter member id: 20130430
Enter book id: 3
Loan is successfully processed
```

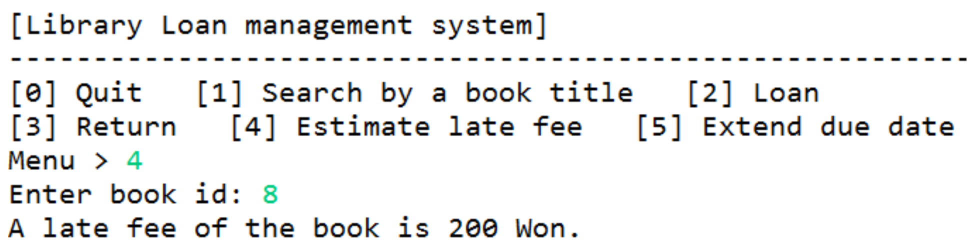
- Print an error message in the catch clause in "try-catch" block if exception occurs
  - String getMessage()
    - ✓ A method in 'Exception' class returns an error message of user-defined exception in stored procedure/function
- Print message if the request is processed successfully

Homework #5 (cont'd)

- Example



```
[Library Loan management system]
---------------------------------------------------------
[0] Quit   [1] Search by a book title    [2] Loan
[3] Return   [4] Estimate late fee    [5] Extend due date
Menu > 4
Enter book id: 100
ORA-20001: The book is not in BookInfo table
ORA-06512: at "JHSEO.GETLATEFEE", line 38
ORA-06512: at line 1
```



```
[Library Loan management system]
---------------------------------------------------------
[0] Quit   [1] Search by a book title    [2] Loan
[3] Return   [4] Estimate late fee    [5] Extend due date
Menu > 4
Enter book id: 8
A late fee of the book is 200 Won.
```

- Print an error message in the catch clause in "try-catch" block if exception occurs
  - String getMessage()
    - ✓ A method in 'Exception' class returns an error message of user-defined exception in stored procedure/function
- Print message if the request is processed successfully

Submission

- **Uploaded file on KLMS**
  - InsertScript.sql : a script that inserts records of Loan, BookInfo, Members table
- **Due**
  - **May. 17 (Sun),** 12:00 p.m. (noon)
  - Delay is not accepted
- **TA info**
  - Junghyuk Seo (Tel : x7830 , email : jhseo@dbserver.kaist.ac.kr)
- **Files to submit**
  - **7 files for problem1 – problem 7 (*.sql)**
    - filenames should be like the following,
      - e.g., ***problem1.sql, problem2.sql, problem3.sql, etc.***
  - Java source code
- **How to submit**
  - Submit to the Assignment #5 board in KLMS system
  - archive all the requested files into [studentID].zip and upload it

Reference

- **Text book (**Database systems - the complete book 2nd edition)
  - Chap 6. SQL
  - Chap 7. Constraints and Triggers
  - Chap 8. Views and Indexes
- **PL/SQL User's Guide and Reference**
  - http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/toc.htm